université de BORDEAUX

Formation CNRS
18 Novembre 2016
Python pour la biologie

Biopython

cgfb
BIOINFORMATIQUE

université de BORDEAUX

SeqRecord class from Bio.SeqRecord

- Allows higher level features such as identifiers and features (as SeqFeature objects) to be associated with the sequence, and is used throughout the sequence input/output interface Bio.SeqIO described later.
- Using richly annotated sequence data, say from GenBank or EMBL files,
- Cover most things to do with the SeqRecord and SeqFeature objects
- Read the SeqRecord wiki page (http://biopython.org/wiki/SeqRecord), and the built in documentation (also online SeqRecord and SeqFeature)

```
>>> from Bio.SeqRecord import SeqRecord
>>> help(SeqRecord)
...
```

The SeqRecord Object from Bio.SeqRecord module

- .seq - The sequence itself, typically a Seqobject.
- .id - The primary ID used to identify the sequence – string
  - (In most cases this is something like an accession number).
- .name – A “common" name/id for the sequence – string
  - (In some cases this will be the same as the accession number, but it could also be a clone name). analogous to the LOCUS id in a GenBank record.
- .description - A human readable description or expressive name for the sequence - string
- .letter_annotations - Holds per-letter-annotations using a (restricted) dictionary of additional information about the letters in the sequence. The keys are the name of the information, and the information is contained in the value as a Python sequence (i.e. a list, tuple or string) with the same length as the sequence itself. This is often used for quality scores (e.g. Section 20.1.6) or secondary structure information (e.g. from Stockholm/PFAM alignment les).

The SeqRecord Object (2)

- .letter_annotations - Holds per-letter-annotations using a (restricted) dictionary of additional information about the letters in the sequence. The keys are the name of the information, and the information is contained in the value as a Python sequence (i.e. a list, tuple or string) with the same length as the sequence itself. This is often used for quality scores (e.g. Section 20.1.6) or secondary structure information (e.g. from Stockholm/PFAM alignment files).
- .annotations - A dictionary of additional information about the sequence. The keys are the name of the information, and the information is contained in the value. This allows the addition of more “unstructured" information to the sequence.
- .features - A list of SeqFeature objects with more structured information about the features on a sequence (e.g. position of genes on a genome, or domains on a protein sequence). The structure of sequence features is described below in Section 4.3.
- .dbxrefs - A list of database cross-references as strings.

Creating a SeqRecord from scratch

- Usually you won't create a SeqRecord “by hand", but instead use Bio.SeqIO to read in a sequence file for you
- Create a SeqRecord

```
>>> from Bio.Seq import Seq
>>> simple_seq = Seq("GATC")
>>> from Bio.SeqRecord import SeqRecord
>>> simple_seq_r = SeqRecord(simple_seq)
```

- Pass the id, name and description to the initialization function or create later

```
>>> simple_seq_r.id
'<unknown id>'
>>> simple_seq_r.id = "AC12345"
>>> simple_seq_r.description = "Made up sequence I wish I could write a paper about"
>>> print(simple_seq_r.description)
Made up sequence I wish I could write a paper about
>>> simple_seq_r.seq
Seq('GATC', Alphabet())
```

Creating a SeqRecord from scratch(2)

- Identifier is very important if you want to output your SeqRecord to a file

```
>>> from Bio.Seq import Seq
>>> simple_seq = Seq("GATC")
>>> from Bio.SeqRecord import SeqRecord
>>> simple_seq_r = SeqRecord(simple_seq, id="AC12345")
```

- The SeqRecord has an dictionary attribute annotations

```
>>> simple_seq_r.annotations["evidence"] = "None. I just made it up."
>>> print(simple_seq_r.annotations)
{'evidence': 'None. I just made it up.'}
>>> print(simple_seq_r.annotations["evidence"])
None. I just made it up.
```

- Working with per-letter-annotations is similar, letter_annotations is a dictionary like attribute which will let you assign any Python sequence (i.e. a string, list or tuple) which has the same length as the sequence

```
>>> simple_seq_r.letter_annotations["phred_quality"] = [40, 40, 38, 30]
>>> print(simple_seq_r.letter_annotations)
{'phred_quality': [40, 40, 38, 30]}
>>> print(simple_seq_r.letter_annotations["phred_quality"])
[40, 40, 38, 30]
```

SeqRecord objects from FASTA files

➔ https://github.com/biopython/biopython/blob/master/Tests/GenBank/NC_005816.fna

```
>gi|45478711|ref|NC_005816.1| Yersinia pestis biovar Microtus ... pPCP1, complete sequence
TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGATCCAGGGGGTAATCTGCTCTCC
...
```

```
>>> from Bio import SeqIO
>>> record = SeqIO.read("NC_005816.fna", "fasta")
>>> record
SeqRecord(seq=Seq('TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGATCCAGG...CTG',
SingleLetterAlphabet()), id='gi|45478711|ref|NC_005816.1|', name='gi|45478711|ref|NC_005816.1|',
description='gi|45478711|ref|NC_005816.1| Yersinia pestis biovar Microtus ... sequence',
dbxrefs=[])
```

```
>>> record.seq
Seq('TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGATCCAGG...CTG', SingleLetterAlphabet())
```

SeqRecord objects from FASTA files (2)

- The first word (after removing the > symbol) is used for both the id and name attributes
- The whole title line (after removing the greater than symbol) is used for the record description

```
>>> record.id
'gi|45478711|ref|NC_005816.1|'
>>> record.name
'gi|45478711|ref|NC_005816.1|'
>>> record.description
'gi|45478711|ref|NC_005816.1| Yersinia pestis biovar Microtus ... pPCP1, complete sequence'
```

- Note that none of the other annotation attributes get populated when reading a FASTA file

```
>>> record.dbxrefs
[]
>>> record.annotations
{}
>>> record.letter_annotations
{}
>>> record.features
[]
```

SeqRecord objects from GenBank files

- https://github.com/biopython/biopython/blob/master/Tests/GenBank/NC_005816.gb
- Contains a single record (i.e. only one LOCUS line) and starts:

```
LOCUS NC_005816 9609 bp DNA circular BCT 21-JUL-2008
DEFINITION Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete
sequence.
ACCESSION NC_005816
VERSION NC_005816.1 GI:45478711
PROJECT GenomeProject:10638
...
```

- Contains a single record (i.e. only one LOCUS line) and starts:

```
>>> from Bio import SeqIO
>>> record = SeqIO.read("NC_005816.gb", "genbank")
>>> record
SeqRecord(seq=Seq('TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGATCCAGG...CTG',
IUPACAmbiguousDNA()), id='NC_005816.1', name='NC_005816',
description='Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence.',
dbxrefs=['Project:10638'])
```

- Automatically assign a more specfic alphabet

```
>>> record.seq
Seq('TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGATCCAGG...CTG', IUPACAmbiguousDNA())
```

SeqRecord objects from GenBank files (2)

- The name comes from the LOCUS line, while the id includes the version sufix. The description comes from the DEFINITION line:

```
>>> record.id
'NC_005816.1'
>>> record.name
'NC_005816'
>>> record.description
'Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence.'
```

- GenBank files don't have any per-letter annotations:

```
>>> record.letter_annotations
{}
```

- Most of the annotations information gets recorded in the annotations dictionary, for example:

```
>>> len(record.annotations)
11
>>> record.annotations["source"]
'Yersinia pestis biovar Microtus str. 91001'
```

- The dbxrefs list gets populated from any PROJECT or DBLINK lines:

```
>>> record.dbxrefs
['Project:10638']
>>> len(record.features)
29
```

SeqFeature objects

- Attempts to encapsulate as much of the information about the sequence as possible
- based on the GenBank/EMBL feature tables
- The key idea about each SeqFeature object is to describe a region on a parent sequence, typically a SeqRecord object
- This region is described with a location object, typically a range between two positions

- .type - This is a textual description of the type of feature (for instance, this will be something like `CDS or `gene').
- .location - The location of the SeqFeature on the sequence that you are dealing with. The SeqFeature delegates much of its functionality to the location object, and includes a number of shortcut attributes for properties of the location:
  - .ref - shorthand for .location.ref - any (different) reference sequence the location is referring to (Usually just None).
  - .ref_db - shorthand for .location.ref_db - specifies the database any identifier in .ref refers to (Usually just None).
  - .strand - shorthand for .location.strand - the strand on the sequence that the feature is located on. For double stranded nucleotide sequence this may either be 1 for the top strand,1 for the bottom strand, 0 if the strand is important but is unknown, or None if it doesn't matter. This is None for proteins, or single stranded sequences.

SeqFeatures funtionalities

- .qualifiers - Python dictionary of additional information about the feature.
  - The key is some kind of terse one-word description of what the information contained in the value is about, and the value is the actual information.
  - For example, a common key for a qualifier might be \evidence" and the value might be \computational (non-experimental)." This is just a way to let the person who is looking at the feature know that it has not be experimentally (i. e. in a wet lab) con rmed. Note that other the value will be a list of strings (even when there is only one string). This is a re ection of the feature tables in GenBank/EMBL files.
- .sub_features - Represent features with complicated locations like `joins' in GenBank/EMBL files.
  - This has been deprecated with the introduction of the CompoundLocation object, and should now be ignored.

Positions and locations

- Position
  - This refers to a single position on a sequence, which may be fuzzy or not. For instance, 5, 20, <100 and >200 are all positions.
- Location
  - A location is region of sequence bounded by some positions. For instance 5..20 (i. e. 5 to 20) is a location
- FeatureLocation object
  - Need start and end coordinates and a strand
- CompoundLocation object
  - Made up of several region
- FuzzyLocation
  - Several types of fuzzy positions, so we have five classes do deal with them

Fuzzy positions

- ExactPosition
  - Represents a position which is specified as exact along the sequence. a number, and you can get the position by looking at the position attribute of the object.
- BeforePosition
  - Represents a fuzzy position that occurs prior to some specified site. In GenBank/ EMBL notation, this is represented as something like ‘<13', signifying that the real position is located somewhere less than 13. To get the specified upper boundary, look at the position attribute of the object.
- AfterPosition
  - Represents a position that occurs after some specified site. This is represented in GenBank as `>13', and like BeforePosition, you get the boundary number by looking at the position attribute of the object.

Fuzzy positions (2)

- WithinPosition
  - Models a position which occurs somewhere between two specified nucleotides. In GenBank/EMBL notation, this would be represented as '(1.5)', to represent that the position is somewhere within the range 1 to 5. To get the information in this class you have to look at two attributes. The position attribute specifies the lower boundary of the range we are looking at, so in our example case this would be one. The extension attribute specifies the range to the higher boundary, so in this case it would be 4. So object.position is the lower boundary and object.position + object.extension is the upper boundary.
- OneOfPosition
  - Occasionally used for GenBank/EMBL locations, this class deals with a position where several possible values exist, for instance you could use this if the start codon was unclear and there where two candidates for the start of the gene. Alternatively, that might be handled explicitly as two related gene features.
- UnknownPosition
  - This class deals with a position of unknown location. This is not used in Gen- Bank/EMBL, but corresponds to the '?' feature coordinate used in UniProt.

Fuzzy positions (3)

- Here's an example where we create a location with fuzzy end points:

```
>>> from Bio import SeqFeature
>>> start_pos = SeqFeature.AfterPosition(5)
>>> end_pos = SeqFeature.BetweenPosition(9, left=8, right=9)
>>> my_location = SeqFeature.FeatureLocation(start_pos, end_pos)
>>> print(my_location)
[>5:(8^9)]
```

- Note that the details of some of the fuzzy-locations changed in Biopython 1.59, in particular for BetweenPosition and WithinPosition you must now make it explicit which integer position should be used for slicing etc.
- For a start position this is generally the lower (left) value, while for an end position this would generally be the higher (right) value

```
>>> my_location.start
AfterPosition(5)
>>> print(my_location.start)
>5
>>> my_location.end
BetweenPosition(9, left=8, right=9)
>>> print(my_location.end)
(8^9)
```

Fuzzy positions (4)

- If you don't want to deal with fuzzy positions and just want numbers, they are actually subclasses of integers so should work like integers:

```
>>> int(my_location.start)
5
>>> int(my_location.end)
9
```

- For compatibility with older versions of Biopython you can ask for the nofuzzy_start and nofuzzy_end attributes of the location which are plain integers:

```
>>> my_location.nofuzzy_start
5
>>> my_location.nofuzzy_end
9
```

```
>>> exact_location = SeqFeature.FeatureLocation(5, 9)
>>> print(exact_location)
[5:9]
>>> exact_location.start
ExactPosition(5)
>>> int(exact_location.start)
5
>>> exact_location.nofuzzy_start
5
```

Keyword « in »

- See if the base/residue for a parent coordinate is within the feature/ location or not ?

```
>>> from Bio import SeqIO
>>> my_snp = 4350
>>> record = SeqIO.read("NC_005816.gb", "genbank")
>>> for feature in record.features:
... if my_snp in feature:
... print("%s %s" % (feature.type, feature.qualifiers.get('db_xref')))
...
source ['taxon:229193']
gene ['GeneID:2767712']
CDS ['GI:45478716', 'GeneID:2767712']
```

- Note that gene and CDS features from GenBank or EMBL files defined with joins are the union of the exons - they do not cover any introns.

Sequence described by a feature or location

- A SeqFeature or location object doesn't directly contain a sequence
- The location describes how to get this from the parent sequence

```
>>> from Bio.Seq import Seq
>>> from Bio.SeqFeature import SeqFeature, FeatureLocation
>>> example_parent = Seq("ACCGAGACGGCAAAGGCTAGCATAGGTATGAGACTTCCTTCCTGCCAGTGCTGAGGAACTGGGAGCCTAC")
>>> example_feature = SeqFeature(FeatureLocation(5, 18), type="gene", strand=-1)
```

- Take the parent sequence, slice it to extract 5:18, and then take the reverse complement

```
>>> feature_seq = example_parent[example_feature.location.start:example_feature.location.end].reverse_complement()
>>> print(feature_seq) AGCCTTTGCCGTC
```

- SeqFeature object has an extract method to take care of all this

```
>>> feature_seq = example_feature.extract(example_parent)
>>> print(feature_seq)
AGCCTTTGCCGTC
```

- The location describes how to get this from the parent sequence

Sequence described by a feature or location (2)

- The length of a SeqFeature or location matches that of the region of sequence it describes

```
>>> print(example_feature.extract(example_parent))
AGCCTTTGCCGTC
>>> print(len(example_feature.extract(example_parent)))
13
>>> print(len(example_feature))
13
>>> print(len(example_feature.location))
13
```

- For simple FeatureLocation objects the length is just the difference between the start and end positions.
- For a CompoundLocation objects, the length is the sum of the constituent regions

```
>>> int(my_location.start)
5
>>> int(my_location.end)
9
```

```
>>> my_location.nofuzzy_start
5
>>> my_location.nofuzzy_end
9
```

```
>>> int(my_location.start)
5
>>> int(my_location.end)
9
```

```
>>> my_location.nofuzzy_start
5
>>> my_location.nofuzzy_end
9
```

Comparison

- SeqRecord objects can be very complex, but here's a simple example

```
>>> from Bio.Seq import Seq
>>> from Bio.SeqRecord import SeqRecord
>>> record1 = SeqRecord(Seq("ACGT"), id="test")
>>> record2 = SeqRecord(Seq("ACGT"), id="test")
>>> record1 == record2
```

- What happens when you try to compare these “identical" records?
- In older versions of Biopython, record1 == record2 would only return true if these variables pointed at the same object in memory.

```
>>> record1 == record2
False
```

- In Biopython 1.67, this will raise a exception

```
>>> record1 == record2
Traceback (most recent call last):
NotImplementedError: SeqRecord comparison is deliberately not implemented. Explicitly compare the attributes of interest.
```

- you should check the attributes you are interested in

```
>>> record1.id == record2.id
True
>>> record1.seq == record2.seq
True
```

References

- Bio.SeqFeature.Reference class
  - Journal
  - Title
  - Authors
- Additionally
  - medline_id
  - pubmed_id
  - a comment

The format method

➔ Give a string containing your record formatted using one of the output le formats supported by Bio.SeqIO, such as FASTA

```
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio.Alphabet import generic_protein
record = SeqRecord(Seq("MMYQQGCFAGGTVLRLAKDLAENNRGARVLVVCSEITAVTFRGPSETHLDSMVGQALFGD" \
+"GAGAVIVGSDPDLSVERPLYELVWTGATLLPDSEGAIDGHLREVGLTFHLLKDVPGLISK" \
+"NIEKSLKEAFTPLGISDWNSTFWIAHPGGPAILDQVEAKLGLKEEKMRATREVLSEYGNM" \
+"SSAC", generic_protein),
id="gi|14150838|gb|AAK54648.1|AF376133_1",
description="chalcone synthase [Cucumis sativus]")
print(record.format("fasta"))
>gi|14150838|gb|AAK54648.1|AF376133_1 chalcone synthase [Cucumis sativus]
MMYQQGCFAGGTVLRLAKDLAENNRGARVLVVCSEITAVTFRGPSETHLDSMVGQALFGD
GAGAVIVGSDPDLSVERPLYELVWTGATLLPDSEGAIDGHLREVGLTFHLLKDVPGLISK
NIEKSLKEAFTPLGISDWNSTFWIAHPGGPAILDQVEAKLGLKEEKMRATREVLSEYGNM
SSAC
```

```
>>> my_location.nofuzzy_start
5
>>> my_location.nofuzzy_end
9
```

Slicing a SeqRecord

- Slice a SeqRecord to give you a new SeqRecord covering just part of the sequence

```
>>> from Bio import SeqIO
>>> record = SeqIO.read("NC_005816.gb", "genbank")
>>> record
SeqRecord(seq=Seq('TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGATCCAGG...CTG',
IUPACAmbiguousDNA()), id='NC_005816.1', name='NC_005816',
description='Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence.',
dbxrefs=['Project:10638'])
>>> len(record)
9609
>>> len(record.features)
41
```

- Any per-letter annotations are also sliced !!
- Any features, completely within the new sequence are preserved (with their locations adjusted).

Slicing a SeqRecord (2)

- ➔ Focus in on the pim gene, YP_pPCP05
- ➔ In genbank file, gene/CDS has location string [4343..4780]
- ➔ In Python counting [4342:4780]

```
>>> print(record.features[20])
type: gene
location: [4342:4780](+)
qualifiers:
Key: db_xref, Value: ['GeneID:2767712']
Key: gene, Value: ['pim']
Key: locus_tag, Value: ['YP_pPCP05']
<BLANKLINE>
>>> print(record.features[21])
type: CDS
location: [4342:4780](+)
qualifiers:
Key: codon_start, Value: ['1']
Key: db_xref, Value: ['GI:45478716', 'GeneID:2767712']
Key: gene, Value: ['pim']
Key: locus_tag, Value: ['YP_pPCP05']
Key: note, Value: ['similar to many previously sequenced pesticin immunity ...']
Key: product, Value: ['pesticin immunity protein']
Key: protein_id, Value: ['NP_995571.1']
Key: transl_table, Value: ['11']
Key: translation, Value: ['MGGGMISKLFCLALIFLSSSGLAEKNTYTAKDILQNLELNTFGNSLSH...']
```

Slicing a SeqRecord (3)

➔ Slice this parent record from 4300 to 4800

```
>>> sub_record = record[4300:4800]
>>> sub_record
SeqRecord(seq=Seq('ATAAATAGATTATTCCAAATAATTTATTTATGTAAGAACAGGATGGGAGGGGGA...TTA',
IUPACAmbiguousDNA()), id='NC_005816.1', name='NC_005816',
description='Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence.',
dbxrefs=[])
>>> len(sub_record)
500
>>> len(sub_record.features)
2
```

➔ Our sub-record just has two features, the gene and CDS entries for YP_pPCP05:

```
>>> print(sub_record.features[0])
type: gene
location: [42:480](+)
qualifiers:
Key: db_xref, Value: ['GeneID:2767712']
Key: gene, Value: ['pim']
Key: locus_tag, Value: ['YP_pPCP05']
<BLANKLINE>
```

Slicing a SeqRecord (3)

```
>>> print(sub_record.features[1])
type: CDS
location: [42:480](+)
qualifiers:
Key: codon_start, Value: ['1']
Key: db_xref, Value: ['GI:45478716', 'GeneID:2767712']
Key: gene, Value: ['pim']
Key: locus_tag, Value: ['YP_pPCP05']
Key: note, Value: ['similar to many previously sequenced pesticin immunity ...']
Key: product, Value: ['pesticin immunity protein']
Key: protein_id, Value: ['NP_995571.1']
Key: transl_table, Value: ['11']
Key: translation, Value: ['MGGGMISKLFCLALIFLSSSGLAEKNTYTAKDILQNLELNTFGNSLSH...']
```

➔ Locations have been adjusted to reflect the new parent sequence!

```
>>> sub_record.annotations
{}
>>> sub_record.dbxrefs
[]
```

Annotations and dbxrefs are omitted from the sub-record !!

```
>>> sub_record.id
'NC_005816.1'
>>> sub_record.name
'NC_005816'
>>> sub_record.description
'Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence.'
```

Record id, name and description are preserved !!

Adding SeqRecord objects

- Add SeqRecord objects together, giving a new SeqRecord
- Any common per-letter annotations are also added
- All the features are preserved (with their locations adjusted)
- Any other common annotation is also kept (like the id, name and description)

```
>>> from Bio import SeqIO
>>> record = next(SeqIO.parse("example.fastq", "fastq"))
>>> len(record)
25
>>> print(record.seq)
CCCTTCTTGTCTTCAGCGTTTCTCC
>>> print(record.letter_annotations["phred_quality"])
[26, 26, 18, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 22, 26, 26, 26, 26,
26, 26, 26, 23, 23]
```

```
>>> int(my_location.start)
5
>>> int(my_location.end)
9
```

Adding SeqRecord objects (2)

➔ Suppose this was Roche 454 data and think the TTT should be only TT

```
>>> left = record[:20]
>>> print(left.seq)
CCCTTCTTGTCTTCAGCGTT
>>> print(left.letter_annotations["phred_quality"])
[26, 26, 18, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 22, 26, 26, 26, 26]
>>> right = record[21:]
>>> print(right.seq)
CTCC
>>> print(right.letter_annotations["phred_quality"])
[26, 26, 23, 23]
```

➔ Now add the two parts together

```
>>> edited = left + right
>>> len(edited)
24
>>> print(edited.seq)
CCCTTCTTGTCTTCAGCGTTCTCC
>>> print(edited.letter_annotations["phred_quality"])
[26, 26, 18, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 22, 26, 26, 26, 26,
26, 26, 23, 23]
```

➔ You can make this shorter

```
>>> edited = record[:20] + record[21:]
```

Adding SeqRecord objects (3) –circular genome

```
>>> from Bio import SeqIO
>>> record = SeqIO.read("NC_005816.gb", "genbank")
>>> record
SeqRecord(seq=Seq('TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGATCCAGG...CTG',
IUPACAmbiguousDNA()), id='NC_005816.1', name='NC_005816',
description='Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence.',
dbxrefs=['Project:10638'])
>>> len(record)
9609
>>> len(record.features)
41
>>> record.dbxrefs
['Project:58037']
>>> record.annotations.keys()
['comment', 'sequence_version', 'source', 'taxonomy', 'keywords', 'references',
'accessions', 'data_file_division', 'date', 'organism', 'gi']
```

➔ Shift the origin like this:

```
>>> shifted = record[2000:] + record[:2000]
>>> shifted
SeqRecord(seq=Seq('GATACGCAGTCATATTTTTTACACAATTCTCTAATCCCGACAAGGTCGTAGGTC...GGA',
IUPACAmbiguousDNA()), id='NC_005816.1', name='NC_005816',
description='Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence.',
dbxrefs=[])
>>> len(shifted)
9609
```

Reverse-complementing SeqRecord objects

SeqRecord object's reverse_complement method

```
>>> from Bio import SeqIO
>>> record = SeqIO.read("NC_005816.gb", "genbank")
>>> print("%s %i %i %i %i" % (record.id, len(record), len(record.features), len(record.dbxrefs), len(record.annotations)))
NC_005816.1 9609 41 1 12
```

```
>>> rc = record.reverse_complement(id="TESTING")
>>> print("%s %i %i %i %i" % (rc.id, len(rc), len(rc.features), len(rc.dbxrefs), len(rc.annotations)))
TESTING 9609 41 0 0
```

```
>>> int(my_location.start)
5
>>> int(my_location.end)
9
```

```
>>> my_location.nofuzzy_start
5
>>> my_location.nofuzzy_end
9
```

```
>>> int(my_location.start)
5
>>> int(my_location.end)
9
```

```
>>> my_location.nofuzzy_start
5
>>> my_location.nofuzzy_end
9
```

Parsing sequences file formats : L'objet SeqRecord

- Bioinformatics work involves dealing with the many types of file formats designed to hold biological data.

- These files are loaded with interesting biological data, and a special challenge is parsing these files into a format so that you can manipulate them with some kind of programming language.

- However, the task of parsing these files can be frustrated by the fact that the formats can change quite regularly, and that formats may contain small subtleties which can break even the most well designed parsers.
- Remember to load module SeqIO using
  - from Bio import SeqIO

Simple FASTA parsing example

➔ If you open the lady slipper orchids FASTA file ls_orchid.fasta (94 records) in your favourite text editor, you'll see that the le starts like this:

```
>gi|2765658|emb|Z78533.1|CIZ78533 C.irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA
CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGGAATAAACGATCGAGT
GAATCCGGAGGACCGGTGTACTCAGCTCACCGGGGGCATTGCTCCCGTGGTGACCCTGATTTGTTGTT
GGG
```

```
>>>from Bio import SeqIO
>>>for seq_record in SeqIO.parse("ls_orchid.fasta", "fasta"):
        print(seq_record.id)
        print(repr(seq_record.seq))
        print(len(seq_record))
gi|2765658|emb|Z78533.1|CIZ78533
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC', SingleLetterAlphabet())
740
[...]
gi|2765564|emb|Z78439.1|PBZ78439
Seq('CATTGTTGAGATCACATAATAATTGATCGAGTTAATCTGGAGGATCTGTTTACT...GCC', SingleLetterAlphabet())
592
```

Simple genbank parsing example

```
>>> for seq_record in SeqIO.parse("ls_orchid.gbk", genbank"):
        print(seq_record.id)
        print(repr(seq_record.seq))
        print(len(seq_record))
Z78533.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC', IUPACAmbiguousDNA())
740
[...]
Z78439.1
Seq('CATTGTTGAGATCACATAATAATTGATCGAGTTAATCTGGAGGATCTGTTTACT...GCC', IUPACAmbiguousDNA())
592
```

➔ Using Python iterator is within a list comprehension (or a generator expression)

```
>>> identifiers = [seq_record.id for seq_record in SeqIO.parse("ls_orchid.gbk", "genbank")]
>>> identifiers
['Z78533.1', 'Z78532.1', 'Z78531.1', 'Z78530.1', 'Z78529.1', 'Z78527.1', ..., 'Z78439.1']
```

➔ Also “swiss” for SwissProt files or “embl” for EMBL text files

➔ See wiki page (http://biopython.org/wiki/SeqIO)

Iterating over the records in a sequence file

➔ The object returned by Bio.SeqIO is actually an iterator which returns SeqRecord objects

```
>>>record_iterator = SeqIO.parse("ls_orchid.fasta", "fasta”)
>>>first_record = next(record_iterator)
>>>print(first_record.id)
>>>print(first_record.description)
```

Z78533.1

```
>>>second_record = next(record_iterator)
>>>print(second_record.id)
>>>print(second_record.description)
```

Z78439.1

```
>>>first_record = next(SeqIO.parse("ls_orchid.gbk", "genbank"))
```

Getting a list of the records in a sequence file

- Access records in any order using Python list data type Using a list
- Much more flexible than an iterator (length of the list) but need more memory (hold all the records in memory at once).

```
>>>records = list(SeqIO.parse("ls_orchid.gbk", "genbank"))
>>>print("Found %i records" % len(records))
```

```
Found 94 records
```

```
>>>last_record = records[-1] #using Python's list tricks
>>>print(last_record.id)
>>>print(repr(last_record.seq))
>>>print(len(last_record))
```

```
Z78439.1
Seq('CATTGTTGAGATCACATAATAATTGATCGAGTTAATCTGGAGGATCTGTTTACT...GCC', IUPACAmbiguousDNA())
592
```

```
>>>first_record = records[0] #remember, Python counts from zero
>>>print(first_record.id)
>>>print(repr(first_record.seq))
>>>print(len(first_record))
```

```
Z78533.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC',
IUPACAmbiguousDNA())
740
```

Extracting data

- As an example of how annotations are stored, we'll look at the output from parsing the first record in the GenBank file Is_orchid.gbk.
- Human readable summary of most of the annotation data for the SeqRecord

```
record_iterator = SeqIO.parse("Is_orchid.gbk", "genbank")
first_record = next(record_iterator)
print(first_record)
```

```
ID: Z78533.1
Name: Z78533
Description: C.irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA.
Number of features: 5
/sequence_version=1
/source=Cypripedium irapeanum
/taxonomy=['Eukaryota', 'Viridiplantae', 'Streptophyta', ..., 'Cypripedium']
/keywords=['5.8S ribosomal RNA', '5.8S rRNA gene', ..., 'ITS1', 'ITS2']
/references=[...]
/accessions=['Z78533']
/data_file_division=PLN
/date=30-NOV-2006
/organism=Cypripedium irapeanum
/gi=2765658
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC', IUPACAmbiguousDNA())
```

➔ .annotations attribute which is just a Python dictionary. Like any Python dictionary, you can easily get a list of the keys and values:

```
print(first_record.annotations)
print(first_record.annotations.keys())
print(first_record.annotations.values())
```

➔ extract a list of the species from the ls orchid.gbk GenBank le. The information we want is held in the annotations dictionary under `source' and `organism':

```
>>> print(first_record.annotations["source"])
Cypripedium irapeanum
```

```
>>> print(first_record.annotations["organism"])
Cypripedium irapeanum
```

➔ In general, `organism' is used for the scientific name (in Latin, e.g. Arabidopsis thaliana), while `source' will often be the common name (e.g. thale cress)

- In general, `organism' is used for the scientific name (in Latin, e.g. Arabidopsis thaliana), while `source' will often be the common name (e.g. thale cress)

```
from Bio import SeqIO
all_species = []
for seq_record in SeqIO.parse("ls_orchid.gbk", "genbank"):
all_species.append(seq_record.annotations["organism"])
print(all_species)
```

```
from Bio import SeqIO
all_species = [seq_record.annotations["organism"] for seq_record in \
SeqIO.parse("ls_orchid.gbk", "genbank")]
print(all_species)
```

```
['Cypripedium irapeanum', 'Cypripedium californicum', ..., 'Paphiopedilum barbatum']
```

Parsing sequences from compressed files

Parsing sequences from the net

Parsing SwissProt sequences from the net

Filtering a sequence file

- large file with many sequences in it (e.g. FASTA le or genes, or a FASTQ or SFF file of reads)

```
from Bio import SeqIO
input_file = "big_file.sff"
id_file = "short_list.txt"
output_file = "short_list.sff"
wanted = set(line.rstrip("\n").split(None,1)[0] for line in open(id_file))
print("Found %i unique identifiers in %s" % (len(wanted), id_file))
records = (r for r in SeqIO.parse(input_file, "sff") if r.id in wanted)
count = SeqIO.write(records, output_file, "sff")
print("Saved %i records from %s to %s" % (count, input_file, output_file))
if count < len(wanted):
print("Warning %i IDs not found in %s" % (len(wanted)-count, input_file))
```

- Let's say the list of IDs is in a simple text file, as the first word on each line.
- This could be a tabular file where the first column is the ID
- Note that we use a Python set rather than a list, this makes testing membership faster

Producing randomised genomes

- Same search on randomised versions of the same genome for statistical analysis
- get it from our website, NC_005816.gb
- read it in as a SeqRecord using the Bio.SeqIO.read() function

```
>>> from Bio import SeqIO
>>> original_rec = SeqIO.read("NC_005816.gb", "genbank")
```

- Python random module, in particular the function random.shuffle

```
>>> import random
>>> nuc_list = list(original_rec.seq)
>>> random.shuffle(nuc_list) #acts in situ!
```

```
>>> from Bio.Seq import Seq
>>> from Bio.SeqRecord import SeqRecord
>>> shuffled_rec = SeqRecord(Seq("".join(nuc_list), original_rec.seq.alphabet),
... id="Shuffled", description="Based on %s" % original_rec.id)
```

- Construct a new SeqRecord with a new Seq object using this shuffled list.

Complete script (version 1)

```
import random
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio import SeqIO
original_rec = SeqIO.read("NC_005816.gb","genbank")
handle = open("shuffled.fasta", "w")
for i in range(30):
nuc_list = list(original_rec.seq)
random.shuffle(nuc_list)
shuffled_rec = SeqRecord(Seq("".join(nuc_list), original_rec.seq.alphabet), \
id="Shuffled%i" % (i+1), \
description="Based on %s" % original_rec.id)
handle.write(shuffled_rec.format("fasta"))
handle.close()
```

Complete script (version 2)

```
import random
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio import SeqIO
def make_shuffle_record(record, new_id):
nuc_list = list(record.seq)
random.shuffle(nuc_list)
return SeqRecord(Seq("".join(nuc_list), record.seq.alphabet), \
id=new_id, description="Based on %s" % original_rec.id)
original_rec = SeqIO.read("NC_005816.gb","genbank")
shuffled_recs = (make_shuffle_record(original_rec, "Shuffled%i" % (i+1)) \
for i in range(30))
handle = open("shuffled.fasta", "w")
SeqIO.write(shuffled_recs, handle, "fasta")
handle.close()
```

Translating a FASTA le of CDS entries

```
from Bio.SeqRecord import SeqRecord
def make_protein_record(nuc_record):
"""Returns a new SeqRecord with the translated sequence (default table)."""
return SeqRecord(seq = nuc_record.seq.translate(cds=True), \
id = "trans_" + nuc_record.id, \
description = "translation of CDS, using default table")
```

```
from Bio import SeqIO
proteins = (make_protein_record(nuc_rec) for nuc_rec in \
SeqIO.parse("coding_sequences.fasta", "fasta"))
SeqIO.write(proteins, "translations.fasta", "fasta")
```

```
>>>first_record = records[0] #remember, Python counts from zero
>>>print(first_record.id)
>>>print(repr(first_record.seq))
>>>print(len(first_record))
```

```
Z78439.1
Seq('CATTGTTGAGATCACATAATAATTGATCGAGTTAATCTGGAGGATCTGTTTACT...GCC', IUPACAmbiguousDNA())
592
```

Making the sequences in a FASTA file upper case

Sorting a sequence file

Simple quality filtering for FASTQ files

Trimming of primer sequences

Trimming off adaptor sequences

Converting FASTQ files

Converting FASTA and QUAL files into FASTQ files